Electronic Notes in Theoretical Computer Science 1 (1995)
URL: http://www.elsevier.nl/locate/entcs/volume1.html 19 pages

Concurrent Semantics for the π-calculus*

Ugo Montanari and Marco Pistore

Computer Science Department, University of Pisa
Corso Italia 40, 56100 Pisa, Italy
{ugo,pistore}@di.unipi.it

Abstract

In this paper we give both operational and abstract concurrent semantics for the π-calculus (a process algebra with the ability of handling channels as messages [23]) and discuss their consistency. For the operational semantics, we map the language into graph rewriting systems, which are already equipped with a concurrent semantics [15,6]; for the abstract semantics we introduce interleaving, partial ordering and mixed ordering observations, define the corresponding bisimulation relations and discuss them.

1 Introduction

The π-calculus is a process algebra with the ability of handling channels (or *names*) as messages, thus modeling agents able to change their neighborhood [23]. The π-calculus has recently raised considerable interest also outside the process algebra community for both theoretical and practical reasons. In fact, name passing is enough for simulating higher order concurrent calculi [30]. Furthermore, name passing and higher order capabilities are instrumental in making formal the basic concepts of common paradigms like object oriented programming [34].

A truly concurrent (or simply concurrent) semantics of a concurrent language aims at expressing, in addition to input-output behavior and temporal dependencies, other informations like causal dependencies of actions and spatial distribution of systems. The extra knowledge can be useful, and sometimes essential, in faster and more thorough specification, testing and validation. Moreover, a concurrent semantics intends to put on formal grounds issues — like the amount of parallelism available in a program — that, while of clear practical relevance, would otherwise be considered as implementation details and would be ineffable in formal terms.

* Work supported in part by Esprit Basic Research project CONFER and working group COMPUGRAPH II and by Progetto Speciale CNR "Specifica ad Alto Livello e Verifica Formale di Sistemi Digitali".

This work aims at extending to the π-calculus certain techniques for defining concurrency already experimented in other process algebras like CCS and at drawing conclusions in terms of the resulting, expected or desired semantics.

Several different notions are often designated as concurrent semantics. *Operational* concurrent models are transition systems equipped with a notion of *abstract computation*. Abstract computations are equivalence classes of computations with the intended meaning that a class contains all the executions, in different order, or in parallel, of the same concurrent events.

Recent results show that operational models can be built axiomatically in a systematic way for a variety of formalisms, like Petri nets [13], phrase structure grammars, term [20] and graph [5] rewriting systems, λ-calculus [19], Horn clauses [7]. Usually, abstract computations have a precise meaning: for instance they are syntactic derivation trees in the case of context free string grammars and nonsequential processes [17] in the case of Petri nets. In the case of process algebras, direct axiomatization is not easy: equivalence of computations is usually defined either directly via the residual method [3] or indirectly by mapping the calculus into one of the concurrent formalisms above, typically Petri nets [12,28,18].

Abstract models introduce a notion of observation. Observations are used to define equivalence of states and agents, typically via bisimulation. When a concurrent semantics is sought, computations rather than transitions must be observed, since the causes of an event must be recovered in the past. Alternatively, a larger transition system can be defined, where a state contains the past computation, or an abstraction of it. The latter technique, which we will refer to as the *unfolding* approach, has been introduced for causal trees [10] and later employed for locality-based semantics [4]. Direct observation of computations leads to the same results [27], with the advantage of employing the same transition system for several different, possibly related observations.

Abstract models do not explicitly reveal parallelism of agents. However a language can be equipped with both an operational and an abstract semantics to fulfill both needs. Of course there is a natural notion of consistency between the two semantics: two computations corresponding to the same abstract computation must have the same observation.

The π-calculus has often been informally explained [21] in term of the chemical analogy of CHAM [1], which implies a concurrent understanding of the model, and in terms of graph reduction [22,29]. However, only recently the concurrent semantics of the π-calculus has been studied in detail [31,2]. In [31], Sangiorgi extends to the π-calculus the locality-based approach of [4]. The interesting observation is that the mechanism employed by the ordinary π-calculus to keep trace of the identity of names is similar to the one used in [4] to identify abstract locations. In [2], Boreale and Sangiorgi extend the above result to the causality-based approach. As it should be clear from the discussion, their approach is abstract and unfolding-based according to our classification.

In this paper we give both operational and abstract concurrent semantics for the π-calculus and discuss their consistency. For the operational semantics,

we follow the approach of mapping the language into a model of computation equipped with a concurrent semantics. The model we choose is graph rewriting based on the double-pushout approach [15].

There are several reasons to choose graph rewriting as our metalanguage. One general reason is that graph rewriting and term graph rewriting look now very promising from several points of view for modeling reduction processes, in particular optimal reduction. Another reason is that graph rewriting systems naturally extend Petri nets (and CHAM) in their ability of describing distributed systems. In fact, nets are just rewriting systems where a state is a *case* or a *marking*, i.e., a set or multiset of places, and rewriting rules are transitions. If we consider labelled graphs, then markings are exactly represented by *discrete* graphs, i.e., graphs with no arcs and up to isomorphism. For instance the multiset $2a \oplus 3b$ is represented by the discrete graph with five nodes, two of them labelled by a and the remaining three labelled by b. The additional information in non-discrete graphs is essentially the adjacency relation between arcs and nodes: in our case it will be very useful in modeling the agents vs. names relation in the π-calculus.

There are more specific, technical advantages in favor of double-pushout graph rewriting. The main one is that all the constructions in that approach are defined via universal diagrams, which identify the results only up to isomorphism. When nodes are names and arcs are agents, *up to isomorphism* essentially means *up to alpha conversion*. Thus graphs can be seen as normal forms for axioms like alpha conversion, associativity and commutativity of parallel composition and others. More interestingly, a rewriting step on graphs can naturally create *new* nodes, i.e., new names in our case. The issue of the identity of a new item in the next derivation steps is directly taken care of by the basic definitions of graph rewriting, without resorting to unnecessary infinite branching, as it is the case in the ordinary definition of the π-calculus. Finally, concurrent aspects of double-pushout graph rewriting have already been considered in the pioneering work by Ehrig and Kreowski in the late seventies. Only recently, however, most concurrency theory available for Petri nets has been extended to graph rewriting systems, including Winskel constructions of event structures and prime algebraic domains [35], extended by [32,6]. Also, the categorical model of computation of [13] with its characterization of arrows as Petri processes, has been extended by [5,9].

The version of the π-calculus we consider does not contain nondeterministic sum and has a form of replication limited to the input prefix. However, our replication construct is powerful enough to express guarded recursion with multiple declarations, which in turn can implement guarded sums. We also omit matching. The advantage is that our restricted version of π-calculus can be given an operational semantics using only nonconditional rewriting rules, which makes discussion and examples simpler.

An interesting result is as follows. The operational semantics above requires production schemata. However, we can show that, if a particular agent is considered, only a finite number of graph productions is required. In comparison, only CCS agents without restriction can be mapped into finite Petri

nets [18]: the mapping is not possible in general [33]. The additional power we have here is due to the superior ability of graphs in handling names.

As we mentioned before, once mapped into a graph rewriting system, π-calculus can be given an operational concurrent semantics using known results [5]. The resulting notion of parallelism is rather interesting. For instance the agent

$$P = (\nu y)(\bar{x}y.0 \mid y(v).0 \mid \bar{z}y.0)$$

has two subagents $\bar{x}y.0$ and $\bar{z}y.0$ both able to create the same new global name. In our semantics they are allowed to extrude in parallel. This is not the case in the semantics of [2] which forces a sequentialization instead [1]. The middle subagent $y(v).0$ needs to wait until y has been extruded. If both previous subagents did it, in our semantics the third subagent picks up nondeterministically the causal dependency of either of them. As another example, consider the agent

$$P = !i(v).(\nu r)\bar{o}r.0$$

which can receive on a global channel i an unbound number of inputs in parallel. To each input, it causally follows the extrusion of a new name on another global channel o. Every event in a pair is in general concurrent with every event in any other pair. Only when the value of an input corresponds to the value of a previous output the two events are sequentialized.

When considering the abstract semantics, we introduce three different observations: interleaving, partial ordering and mixed ordering. Then an ordinary bisimulation relation is defined on agents, which is independent from the observation. We emphasize that the standard process algebra theory and tools can be applied at this point.

As expected, the equivalence induced by interleaving observations is the ordinary early observational equivalence. Mixed ordering observations produce an equivalence finer than both interleaving and partial ordering. The surprising result here is that for the π-calculus the partial ordering observation does not induce a finer equivalence than the interleaving observation (as it is the case [11] for CCS), but rather an incomparable equivalence. The above result becomes relevant when we notice that partial ordering observations are consistent with the operational concurrent semantics we defined via graph rewriting, while both interleaving and mixed ordering observations are not.

[1] As discussed in the conclusions, we believe to be able to express the alternative semantics of [2] with little effort. We do not see, instead, how our more parallel semantics could be easily modeled in the style of [2]. We thank Davide Sangiorgi for showing us this example.

2 Syntax and operational semantics

2.1 Syntax and ordinary semantics

Given an infinite set of *names* $\mathcal{N}$ ($a, b, \ldots, x, y, z \in \mathcal{N}$), the π-calculus *agents* over $\mathcal{N}$ are defined by the syntax:

$$P ::= \alpha.P \mid P_1|P_2 \mid (\nu x)P \mid 0$$

where the *prefixes* α are defined by the syntax:

$$\alpha ::= \bar{x}y \mid x(y) \mid !x(y).$$

The occurrences of y in $x(y).P$, $!x(y).P$ and $(\nu y)P$ are bound; free and bound names are then defined as usual: we indicate with $fn(P)$ and $bn(P)$ the free names and the bound names of P respectively.

The *actions* an agent can perform are defined by the following syntax:

$$\mu ::= \tau \mid \bar{x}y \mid xy \mid \bar{x}(z);$$

x and y are free names of μ, whereas z is a bound name; $fn(\mu)$ and $bn(\mu)$ are respectively the free and bound names of μ and $n(\mu) = fn(\mu) \cup bn(\mu)$.

The *transitions* are defined by the axiom schemata and the inference rules of Table 1.

OUT $\bar{x}y.P \xrightarrow{\bar{x}y} P$	IN $x(y).P \xrightarrow{xz} P\{z/y\}$	!IN $!x(y).P \xrightarrow{xz} !x(y).P \mid P\{z/y\}$
PAR $\dfrac{P \xrightarrow{\mu} P'}{P\mid Q \xrightarrow{\mu} P'\mid Q}$ if $bn(\mu)\cap fn(Q)=\emptyset$		COM $\dfrac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{xy} Q'}{P\mid Q \xrightarrow{\tau} P'\mid Q'}$
CLOSE $\dfrac{P \xrightarrow{\bar{x}(y)} P' \quad Q \xrightarrow{xy} Q'}{P\mid Q \xrightarrow{\tau} (\nu y)(P'\mid Q')}$ if $y \notin fn(Q)$		RES $\dfrac{P \xrightarrow{\mu} P'}{(\nu x)P \xrightarrow{\mu} (\nu x)P'}$ if $x \notin n(\mu)$
OPEN $\dfrac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(z)} P'\{z/y\}}$ if $x \neq y, z \notin fn((\nu y)P')$		

Table 1 Ordinary operational semantics

We have omitted the symmetric rules for PAR, COM and CLOSE. These transitions define an *early* operational semantics for the π-calculus; this version was first introduced in [24], but we have slightly simplified it, following in part the style proposed in [30,31,2] for polyadic π-calculus.

Example 2.1 A possible computation of a π-calculus agent P is the following:

$$\begin{aligned} P = (\nu w)(\nu z)(!x(y).\bar{z}y.0 \mid \bar{x}w.z(v).0) &\xrightarrow{\bar{x}(w)} (\nu z)(!x(y).\bar{z}y.0 \mid z(v).0) \\ &\xrightarrow{xx} (\nu z)(!x(y).\bar{z}y.0 \mid \bar{z}x.0 \mid z(v).0) \\ &\xrightarrow{\tau} (\nu z)(!x(y).\bar{z}y.0 \mid 0 \mid 0). \end{aligned}$$

Definition 2.2 A relation $\mathcal{R}$ over agents is an *early simulation* iff, given $P\mathcal{R}Q$:

- whenever $P \xrightarrow{\mu} P'$ with $bn(\mu) \cap fn(P, Q) = \emptyset$, then $Q \xrightarrow{\mu} Q'$ and $P'\mathcal{R}Q'$.

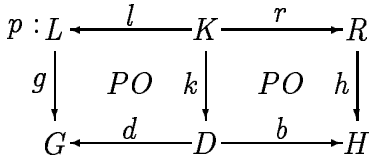


Fig. 1. A double-pushout diagram

Relation $\mathcal{R}$ is an *early bisimulation* iff $\mathcal{R}$ and $\mathcal{R}^{-1}$ are early simulations. Two agents P and Q are *early bisimilar* ($P \sim_e Q$) iff $P\mathcal{R}Q$ for some early bisimulation $\mathcal{R}$.

2.2 Graph rewriting systems

The graphs we consider here are rather *labelled hypergraphs*, i.e., their (hyper)arcs connect not necessarily a pair of nodes, but a tuple of them. Moreover arcs are typed, i.e., there exists a *labelling function* from the arcs to a set of arc types; we assume that each arc type has a fixed rank, indicating how many nodes must be connected to an arc of such type. A *morphism* between two graphs is then a pair of functions between nodes and arcs of the two graphs which respect connections and arc types.

A *production* $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ is characterized by a pair of injective graph morphisms l and r having as source the *gluing graph* K and as destination the graphs L and R respectively (they are named *left hand* and *right hand side* of the production).

The role of the gluing graph K and of the two morphisms is to identify the part of the left hand side L which is preserved by the rule, namely the image of K in L which becomes the image of K in R. This part can be understood as being read but not consumed during a rewriting step. In the concurrent semantics of graph rewriting, two rule applications sharing no elements or only elements in their gluing graphs can be applied in parallel, i.e., readers can proceed in parallel, while a reader and a writer, or two writers, must be sequentialized. The ability of distinguishing between reading and consuming resources (rather than between permanent and consumable resources as in linear logic) gives a particularly expressive power to graph rewriting. In this work, gluing graphs will contain both processes to be replicated and information about names being unrestricted, which, once established, can be used several times.

In order to apply a production $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ to a graph G yielding H (in brief $G \Rightarrow_p H$) there must be an *occurrence* of L in G, i.e., a graph morphism $g : L \rightarrow G$, and H must be the graph obtained as the result of the double-pushout construction of Figure 1, where all arrows are graph morphisms.

We now comment on the double-pushout construction, which is the core of the approach[2]. Let us assume we have chosen a production p and an

[2] The double-pushout is a general construction for rewriting systems: changing the base category, it can be used to describe rewritings in structures different than graphs; e.g., if the category of labelled sets is used as base, a double-pushout diagram corresponds to a

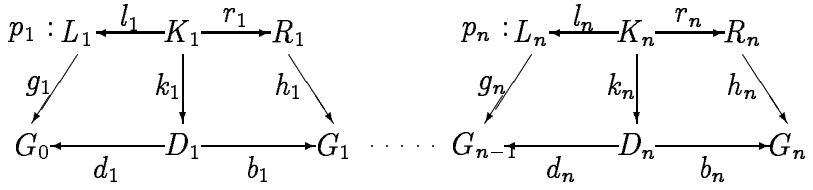


Fig. 2. A graph derivation

occurrence in the given graph of the left hand side L, i.e., we have fixed the morphism g. The first step is to construct the *pushout complement* D with the two morphisms k and d. Informally, one has to cover graph G with two subgraphs L and D whose intersection is K. It is possible to show that this is possible if and only if: i) morphism g is injective when restricted to $L - l(K)$ (i.e., consumption is resource conscious); and ii) if a node of G is in $g(L - l(K))$, i.e., it is consumed, so are all the arcs connected to it (this is necessary to avoid dangling arcs in D). Moreover, the result is unique, up to isomorphism. Once D (i.e., the part of the graph not affected by the rule) is available, then the result H can be simply obtained by taking the union of D and of the right hand side R, with the proviso that elements in the image of K are identified. Again, the result is unique up to isomorphism.

A *graph rewriting system* is then characterized as consisting of a set of productions and of an initial graph. A *derivation* $G_0 \Rightarrow^* G_n$ is a sequence of double-pushouts as schematized in Figure 2.

2.3 Concurrent operational semantics

In this section we introduce our representation, based on graph rewriting, of π-calculus agents and of their evolution. Names will be nodes, while arcs will be *sequential processes* and *pebbles*. In an agent, sequential processes will correspond to unguarded occurrences of subagents of the form $S ::= \alpha.P$ which are called *sequential agents*. A pebble is a special arc of rank one and expresses the fact that the node it is connected to is an unrestricted name.

In order to denote the graph corresponding to a whole agent, we need to introduce a refined notion of graph and certain operations on graphs. They are essentially the same of the CHARM abstract machine [8], which moreover is equipped with an axiomatization of graphs and of the double-pushout construction. In this work we have chosen not to introduce the CHARM machine and to refer to the more intuitive graph constructions.

Partially abstract graphs have a specified subgraph, called the *global* part, where nodes and arcs keep their identities, while the rest of the graph, the *local* part, (which is not necessarily a subgraph) is defined only up to isomorphism. A partially abstract graph can be understood as an ordinary graph simply by removing the distinction between global and local part.

In our case, the global part will consist only of nodes corresponding to free names, while the local part will model the rest. The identity of a node in the

transition of a P/T Petri Net (or, more generally, of a Contextual Petri Net [26]).

global part will just be the corresponding name.

To each arc is assigned a label (or type) which is something similar to a sequential process. More precisely, given an arc corresponding to the sequential process S, its associated type is defined as

$$\lfloor S \rfloor = S\{\bullet/x_1, \ldots, \bullet/x_n\}$$

where $\{x_i\} = fn(S)$ and $\bullet$ is a special mark. The rank of $\lfloor S \rfloor$ will be the number of marks in it. Pebbles will all have the same type, of rank one.

The base graphs we will use are: *i*) the graph without arcs and nodes, denoted by 0; *ii*) the graph consisting of a global node x and a pebble on it, denoted by $\downarrow x$; *iii*) the graph corresponding to the sequential agent S. It has as global nodes all the free names of S and just one arc, with the associated type $\lfloor S \rfloor$ of S. The arc has as many connections as there are occurrences of $\bullet$ in $\lfloor S \rfloor$. The ith connection binds node x iff the ith occurrence [3] of $\bullet$ in $\lfloor S \rfloor$ is an occurrence of name x in S. This graph will be denoted by S.

The following operations are defined over graphs:

- *composition* $G = G_1 \oplus G_2$: graph G is obtained by coalescing graphs G_1 and G_2, where the global nodes of G_1 and G_2 with the same identity are identified;
- *restriction* $G = (x_1, \ldots, x_n)G'$: graph G is the same as G', but nodes $\{x_i\}$, if any, now become local;
- *substitution* $G = G'\{x_1/y_1, \ldots, x_n/y_n\}$, where y_i are all distinct: this operation changes the identity of the global nodes y_i of G'.

We now define inductively how to map π-calculus agents into partially abstract graphs.

Definition 2.3 The *unwinding* function from agents to partially abstract graphs is defined by induction over the structure of agents as follows:

$$\begin{aligned} unw(\alpha.P) &= \alpha.P \\ unw(P_1|P_2) &= unw(P_1) \oplus unw(P_2) \\ unw((\nu x)P) &= (x)unw(P) \\ unw(0) &= 0 \end{aligned}$$

Partial graphs will be useful in defining the productions below. However, the states of the system will be represented simply by graphs, in order to apply the ordinary theory of double-pushout graph rewriting. The initial state of the computation will be represented as follows.

Definition 2.4 The graph corresponding to the π-calculus agent P and to the name tuple $M = \langle x_1, \ldots, x_n \rangle$, where the names in M are all distinct and contain $fn(P)$, is the ordinary graph $unw(P) \oplus \downarrow x_1 \oplus \cdots \oplus \downarrow x_n$; we will denote this graph by G_P^M.

Example 2.5 The graph corresponding to agent $P = (\nu w)(\nu z)(!x(y).\bar{z}y.0 \mid$

[3] The occurrences of the free names of S must be ordered in some fixed way, e.g., corresponding to the anticipate traversal of the term.

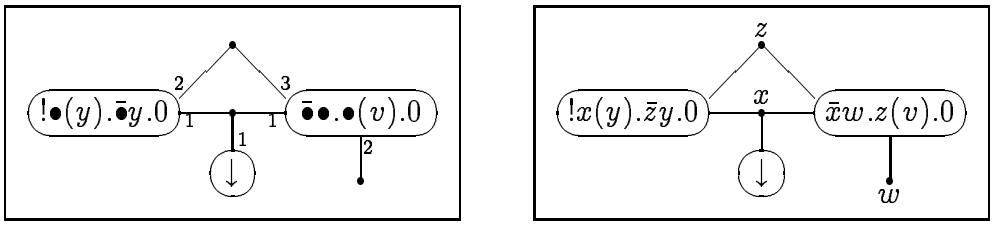


Fig. 3. The graph for agent $P = (\nu w)(\nu z)(!x(y).\bar{z}y.0 \mid \bar{x}w.z(v).0)$ of Example 2.1

$$\text{OUT: } \bar{x}y.P \underset{\downarrow x}{\longrightarrow} unw(P) \oplus \downarrow y$$

$$\text{IN1: } x(y).P \underset{\downarrow x}{\longrightarrow} (y)(unw(P) \oplus \downarrow y) \qquad \text{IN2: } x(y).P \underset{\downarrow x \oplus \downarrow z}{\longrightarrow\!\!\!\!\!\!-} unw(P)\{z/y\}$$

$$\text{!IN1: } \underset{!x(y).P \oplus \downarrow x}{\longrightarrow\!\!\!\!\!\!\!\!-} (y)(unw(P) \oplus \downarrow y) \qquad \text{!IN2: } \underset{!x(y).P \oplus \downarrow x \oplus \downarrow z}{\longrightarrow\!\!\!\!\!\!\!\!\!\!-} unw(P)\{z/y\}$$

$$\text{SYNC: } \bar{x}z.P \oplus x(y).Q \rightarrow unw(P) \oplus unw(Q)\{z/y\}$$

$$\text{!SYNC: } \bar{x}z.P \underset{!x(y).Q}{\longrightarrow\!\!\!\!\!\!-} unw(P) \oplus unw(Q)\{z/y\}$$

Table 2 Graph productions for the π-calculus

$\bar{x}w.z(v).0$) of Example 2.1 (and to the global name x) is represented in Figure 3; on the left we show the graph as we defined it, while on the right we see a more handy representation, which we will use from now on.

The productions for the π-calculus are defined by the production schemata of Table 2. They have the form $I \underset{C}{\rightarrow} O$. If we instantiate in a production schema the agent variables P and Q, then I, C and O are three partially abstract graphs. They define a graph production $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ with $K = C \oplus g(I)$, $L = I \oplus K$ and $R = O \oplus K$, being $g(G)$ the global nodes of G, and where the morphisms are the obvious inclusions.

We now comment on the productions in Table 2. There is one rule of output. In order to apply it, two resources are needed: the agent $\bar{x}y.P$ and the pebble $\downarrow x$ which makes sure that x is unrestricted: the former is consumed and the latter is read. As we mentioned in the introduction, we treat all outputs as extrusions, and thus the pebble $\downarrow y$ is always added to the resulting graph. Notice that the agent P is transformed into a graph by the unwinding function, and thus its distributed structure is made explicit. Input rule IN1 works quite the same way, except that now a new node y is created and used to accommodate the extra connection of P. Rule IN2 accounts for the possibility of inputing a name z which already exists. Notice that it must be unrestricted. Finally, synchronization rule SYNC does not require any context. The remaining rules just duplicate the rules for input and synchronization in the case the input prefix involved is the one with replication. The only difference is that the input agent appears in the gluing graph of the production, i.e., it is read but not consumed.

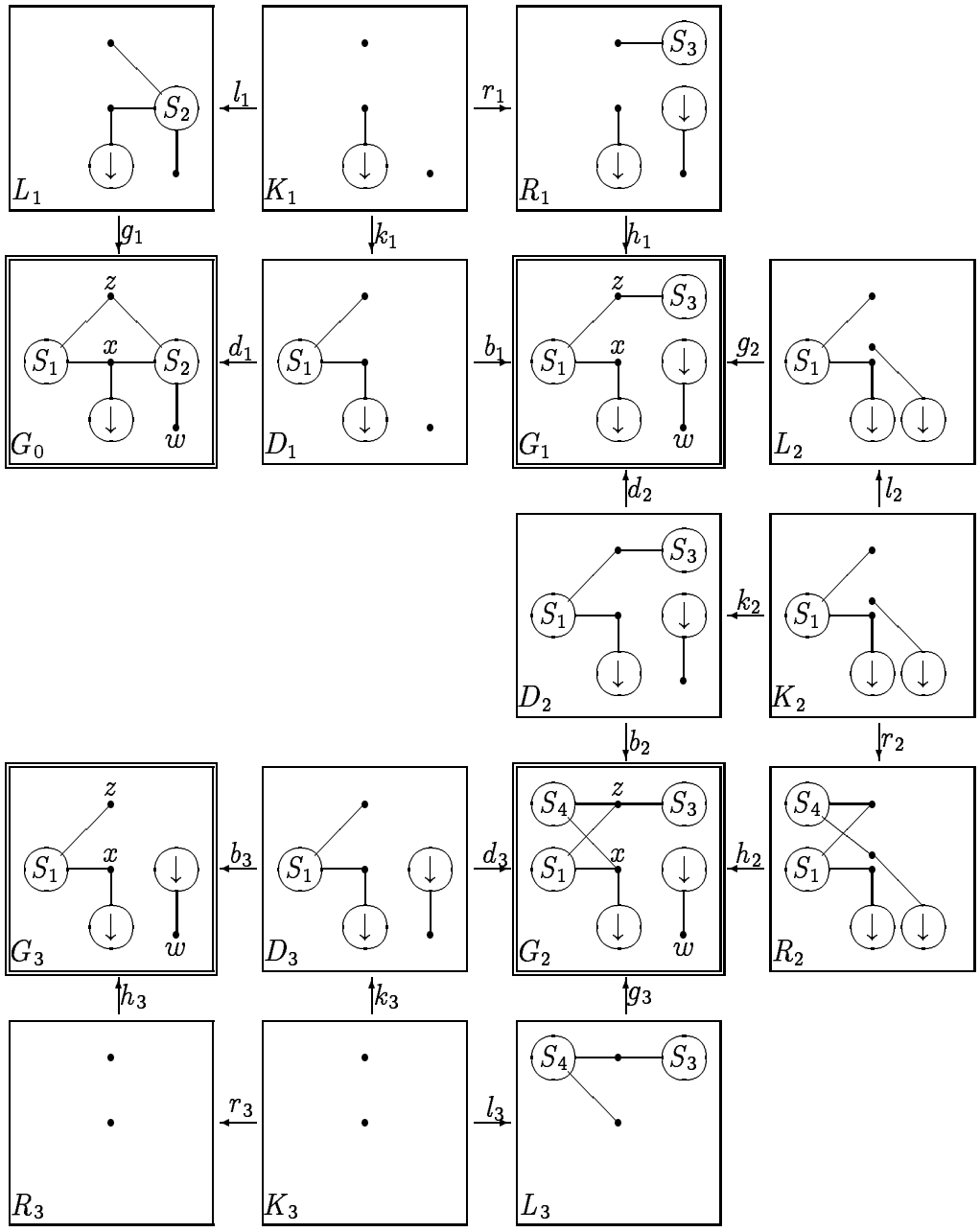
Example 2.6 In Figure 4 we see the same computation of Example 2.1, now represented as a three step double-pushout graph derivation; to represent how arcs and nodes of two graphs are related by a morphism, we have let them occupy the same positions in the squares containing the two graphs. In the first step an instantiation of OUT is used ($\bar{x}w.z(v).0 \underset{\downarrow x}{\longrightarrow} z(v).0 \oplus \downarrow w$): notice how, going from graph G_0 to graph G_1, the context arc $\downarrow x$, which is present in K_1, is unchanged; the arc $S_2 = \bar{x}w.z(v).0$, present only in L_1, is removed and the new arcs $S_3 = z(v).0$ and $\downarrow w$, all present only in R_1, are created; the arc $S_1 = !x(y).\bar{z}y.0$, finally, is not involved in the rewriting. In the second step !IN2 is applied ($\xrightarrow[!x(y).\bar{z}y.0 \oplus \downarrow x \oplus \downarrow x']{} \bar{z}y.0\{x'/y\}$); notice that graph morphism g_2 is not injective, since the same global name x (and its pebble) is used as both the subject (x in the rule) and the object (x') of the input. In the third step an instantiation of rule SYNC is used ($z(v).0 \oplus z(y).0 \longrightarrow 0$).

Once mapped into a graph rewriting system, π-calculus can be given an operational concurrent semantics using known results. We now shortly describe them. As explained in the introduction, the aim is to define suitable equivalence classes of computations. In the double-pushout approach, computations are diagrams in some category of graphs. A first obvious equivalence is thus to define derivations only up to isomorphism. This abstraction essentially corresponds in our case to allow alpha conversion for the names employed in a computation.

A second quotient is obtained by applying the *shift construction*, due to Ehrig and Kreowski [15]. Given a diagram corresponding to a two step derivation where the applied productions are *sequential independent* — i.e., the images in the intermediate graph of the right hand side of the first production and of the left hand side of the second overlap at most on the images of the gluing graphs — the shift construction builds a unique diagram where the two productions are applied in the reverse order. Derivations which can be obtained by repeated application of the shift construction are shift equivalent.

Detailed analysis shows that the two abstractions can be combined without harm only in the case of *safe* systems, i.e., when all the graphs which can be generated have no endomorphisms different from the identity. However, also in the general case a fully satisfactory category can be built by restricting in a suitable way the notion of isomorphism between derivations [5].

It is interesting to realize that the arrows of this category can be characterized as *graph processes* [9]. The construction which, given a derivation, builds its associated graph process, is rather intuitive. It first builds a *common graph* by taking the colimit of the diagram corresponding to the derivation, and then adds as many events as steps in the derivation, with the obvious causal links. In Definition 3.1 we present essentially this construction, but without using colimits, and already abstracting the result with respect to the syntactic identity of agents.



where: $S_1 = !x(y).\bar{z}y.0 \quad S_2 = \bar{x}w.z(v).0$
$S_3 = z(v).0 \quad S_4 = \bar{z}y.0$

Fig. 4. A graph derivation for the agent $P = (\nu w)(\nu z)(!x(y).\bar{z}y.0 \mid \bar{x}w.z(v).0)$ of Example 2.1

2.4 Finiteness

If we are interested in the evolution of a given agent, we need only a finite number of instantiations of the production schemata in Table 2.

Theorem 2.7 *The evolution of any agent P can be described by a finite rewriting system.*

Proof (Sketch) The proof has two steps: the first shows that any sequential process in a graph reachable from G_P^M has the type of some subagent of P. Thus the possible types are finite. The second step shows that the productions in Table 2 that can be applied to graphs with arcs typed on a finite set are finitely many. □

3 Abstract concurrent semantics

3.1 Observations

Derivations keep a lot of low-level details and it is useful to extract from them only the interesting information; this is obtained by defining a set of *observations* and an *observation function* that associates to every derivation its meaning. In this section we will consider particular observations that are interesting for the study of the causal semantics of the π-calculus; other aspects can be considered introducing other observations (see the concluding remarks).

To each production p in Table 2 we associate a label $l(p)$, indicating the action it represents. We have $l(OUT) = output$, $l(IN1) = l(IN2) = l(!IN1) = l(!IN2) = input$ and $l(SYNC) = l(!SYNC) = tau$. Moreover we can distinguish a tuple of nodes $c(p)$ of the right hand side of the production p, corresponding to the names involved in the action: this tuple is empty for *tau*-productions and is the subject-object pair for *input*- and *output*-productions. So $c(OUT) = c(IN1) = c(!IN1) = \langle x, y\rangle$, $c(IN2) = c(!IN2) = \langle x, z\rangle$ and $c(SYNC) = c(!SYNC) = \langle\rangle$.

Definition 3.1 Given a derivation D (see Figure 2) with $G_0 = G_P^M$, let us consider the sets $E = \{e_1, \ldots, e_n\}$ (where e_i represents the event corresponding to the i-th derivation step) and $B = \{\langle a, i\rangle \mid a \in |G_i|\}$, where $|G|$ is the set of arcs and nodes of the graph G. The *dependence relation* $\sqsubseteq$ between events and graphs elements is the smaller partial ordering such that the following rules hold and $\equiv$ is an equivalence relation:

- $a \in |D_i|$ implies $\langle d_i(a), i-1\rangle \equiv \langle b_i(a), i\rangle$;
- $a \in |L_i|$ implies $\langle g_i(a), i-1\rangle \sqsubseteq e_i$;
- $a \in |R_i|, a \notin |r_i(K_i)|$ implies $e_i \sqsubseteq \langle h_i(a), i\rangle$;
- $a \equiv b \sqsubseteq c \equiv d$ implies $a \sqsubseteq d$.

The *mixed ordering* of D is defined as $mo(D) = \langle E, \sqsubseteq_E, \leq, l, c, N, s\rangle$, where:

- $\sqsubseteq_E = \sqsubseteq \cap (E \times E)$;
- $\leq \subseteq E \times E$ is such that $e_i \leq e_j$ iff $i \leq j$;
- $l : E \to \{tau, input, output\}$ is the *labelling function*: $l(e_i) = l(p_i)$;
- $c : E \to N^*$ is the *connection function*: if $c(p_i) = \langle n_1, \ldots, n_m\rangle$ then $c(e_i) = \langle \langle h_i(n_1), i\rangle/_\equiv, \ldots, \langle h_i(n_m), i\rangle/_\equiv\rangle$;
- $N = c(E) = \bigcup_{i\in E} c(e_i)$;
- s is a partial function $N \to \mathcal{N}$ returning the *syntactic identity* of the global

nodes of the initial graph: $s(\langle x, 0\rangle|_{\equiv}) = x$ if x is a global node of $G_0 = G_P^M$ (i.e., a node corresponding to a name of the tuple M).

Moreover the *total ordering* of D is defined as $to(D) = \langle E, \leq, l, c, N, s\rangle$ and *the partial ordering* of D as $po(D) = \langle E, \sqsubseteq_E, l, c, N, s\rangle$.

The structures above are considered up to isomorphism, where an isomorphism between structures is a pair of isomorphisms between events and between nodes that respects the orderings of the events, the labelling and connection functions and the identity of the global nodes.

Let us comment on the construction. For every step in the derivation, an event is generated. Moreover, nonintersecting copies of all the intermediate graphs G_i are created. Items are then identified according to graph morphisms, and causal links are added from all the items in L_i to the event e_i and from it to the items in $R_i - r(K_i)$. Finally, the causal relation is made transitively closed and all arcs and unnecessary nodes are erased. Each event is connected to the nodes (names) appearing in the corresponding action. The nodes which appear also in the initial graph and which are global keep their syntactic identity. Besides information about the action performed, partial ordering observations contain only the causal ordering $\sqsubseteq$, while mixed ordering observations contain also the total ordering $\leq$ in which the events are generated. Interleaving observations contain only the latter.

Since no production for the π-calculus has isolated nodes in its left hand side, the nodes can be forgotten without losing any dependency between events: so in Definition 3.1 we can redefine $|G|$ as corresponding only to the arcs of the graph G. We use this version of the construction in the following example.

Example 3.2 In Figure 5 the dependence relation is represented corresponding to the derivation of Figure 4. In Figure 6 we show the corresponding mixed ordering; notice that two names are involved and only one of them, x, has a syntactic identity.

3.2 Observation equivalences

Given an observation function o on derivations, we define the usual bisimulation relation on derivations and graphs, which induces a corresponding observation equivalence on agents. We then apply the construction to the previously defined observations *mo*, *to* and *po*.

Definition 3.3 Let o be an observation function over derivations; a relation $\mathcal{R}$ over derivations is an *o-simulation* iff, given $D\mathcal{R}E$, where $D : G \Rightarrow^* G'$ and $E : H \Rightarrow^* H'$:

- for all derivations $D' : G \Rightarrow^* G' \Rightarrow G''$ there is some derivation $E' : H \Rightarrow^* H' \Rightarrow H''$ with $o(D') = o(E')$ and $D'\mathcal{R}E'$.

Relation $\mathcal{R}$ is an *o-bisimulation* iff $\mathcal{R}$ and $\mathcal{R}^{-1}$ are o-simulations. Two graphs G and H are *o-bisimilar* ($G \sim_o H$) iff $(G \Rightarrow^0 G)\mathcal{R}(H \Rightarrow^0 H)$ for some o-

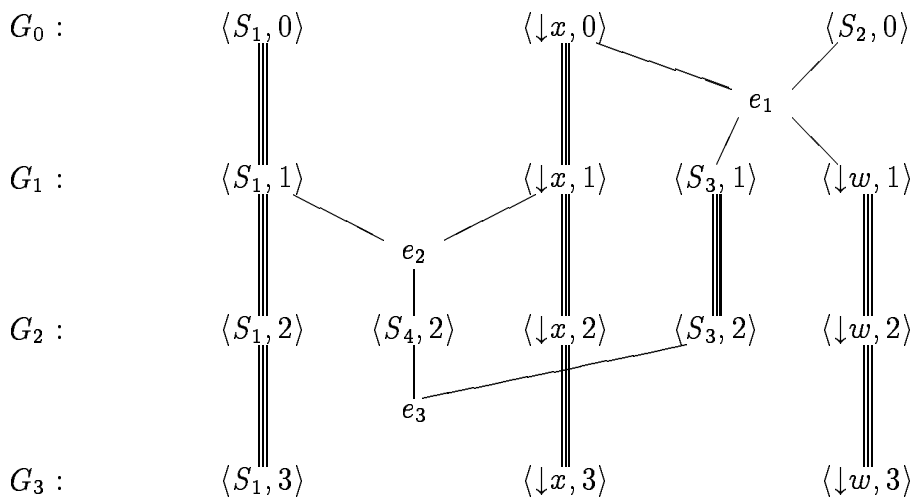


Fig. 5. Dependence relation for the derivation of Figure 4

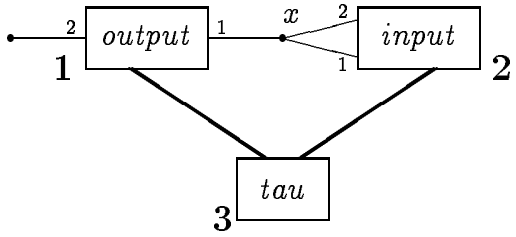


Fig. 6. Mixed ordering observation for the derivation of Figure 4

bisimulation $\mathcal{R}$ ($G \Rightarrow^0 G$ represents the zero-steps derivation from G).

By using the observations *to*, *mo* and *po* of Definition 3.1, we obtain the equivalences over graphs $\sim_{to}$, $\sim_{mo}$ and $\sim_{po}$. From these we can derive the corresponding equivalences over agents.

Definition 3.4 Given two agents P and Q, they are *total ordering equivalent* (resp. *partial ordering, mixed ordering equivalent*) iff, for all tuples M that contain the free names of P and Q, $G_P^M \sim_{to} G_Q^M$ (resp. $G_P^M \sim_{po} G_Q^M$, $G_P^M \sim_{mo} G_Q^M$).

Proposition 3.5 *The relations over agents $\sim_{to}$, $\sim_{po}$ and $\sim_{mo}$ are equivalences.* □

Theorem 3.6 *Given two agents P and Q, $P \sim_e Q$ iff $P \sim_{to} Q$.* □

This theorem shows the correspondence between the ordinary early observational equivalence and the total ordering equivalence obtained with our graph construction, showing that this is an *interleaving* equivalence; the proof follows the style of [27], where two definitions of location equivalence are related.

We can now discuss the resulting equivalences. Of course the equivalence based on mixed orderings is finer than the others. In fact, the more detailed the observation, the finer the equivalence. The following examples show that

the inclusion is strict, and that interleaving and partial ordering equivalences are incomparable.

Theorem 3.7 *Given two agents P and Q, if $P \sim_{mo} Q$ then $P \sim_{to} Q$ and $P \sim_{po} Q$.* □

As one could expect, there are interleaving bisimilar agents which aren't partial ordering bisimilar: for instance consider the agents $\bar{x}x.\bar{x}x.0$ and $\bar{x}x.0|\bar{x}x.0$.

However, the surprising fact is that the converse is also true, i.e., it doesn't hold that if $P \sim_{po} Q$ then $P \sim_{e} Q$. Let us consider the agents:

$$P = (\nu y)(\nu z)(\nu c)(\bar{x}y.\bar{c}y.0 \mid c(w).\bar{w}w.0 \mid \bar{x}z.\bar{c}z.0);$$
$$Q = (\nu y)(\nu z)(\nu c)(\bar{x}y.\bar{c}y.0 \mid c(w).\bar{w}w.0 \mid \bar{x}z.0).$$

Both agents can extrude local names y and z by means of outputs $\bar{x}y$ and $\bar{x}z$. However, P can do an output on y *or* on z (via synchronization over c), whereas Q can do the output *only* on y. It is thus clear that, after the two extrusions, P cannot interleaving-simulate Q, since, no matter which is the order of the extrusions, there is one extruded channel on which Q cannot do the output while P can. If we consider the partial ordering semantics instead, the two names are extruded in parallel and are actually indistinguishable, since they have exactly the same history. Thus, no matter which is the channel used by P, Q can match it [4].

The interesting fact about partial ordering equivalence is that, of the three relations we considered, it is the only one consistent with the concurrent operational semantics we introduced in Subsection 2.3. In fact, if we consider the agent $\bar{x}x.0|\bar{y}y.0$, it is immediate to see that both its derivations leading to $0|0$ have the same partial ordering observation, i.e., two concurrent output events on the channels x and y. Of course the two computations have different interleaving and mixed ordering observations instead, since the two events are in a different total ordering in the two derivations. On the other hand, it is easy to see that the two events are completely independent and thus the shift construction (and any other concurrent operational semantics we can think of) identifies the two derivations. Therefore interleaving and mixed ordering abstract semantics assign different observations to derivations which are identified by the concurrent operational semantics. Instead, it is easy to see that all the derivations identified by the concurrent semantics have the same partial ordering observation. This discussion makes clear that a concurrent abstract semantics is not necessarily implementable in parallel: for instance, assigning to every event a timestamp, i.e., a distinctive number which increases with time, clearly assumes some centralized mechanism which is incompatible with

[4] The example above is not a counterexample yet, since P and Q are not even partial ordering equivalent. The actual counterexample follows the same concept: the agents

$$P = (\nu y)(\nu z)(\nu a)(\nu c)(\bar{x}y.\bar{a}a.\bar{c}y.0 \mid c(w).\bar{w}w.0 \mid \bar{x}z.a(v).\bar{c}z.0),$$
$$Q = (\nu y)(\nu z)(\nu a)(\nu c)(\bar{x}y.\bar{a}a.\bar{c}y.0 \mid c(w).\bar{w}w.0 \mid \bar{x}z.a(v).0)$$

are partial ordering equivalent but not interleaving equivalent.

a concurrent implementation. This criticism applies to all the abstract concurrent semantics based on mixed orderings, and in general to all the semantics where the causality/locality links include the temporal links. We understand this is the case also of [31,2].

4 Concluding remarks

In the paper we presented operational and abstract concurrent semantics for a simplified version of the π-calculus. Guarded sum, matching and full replication require graph inference rules, but little additional complexity is added; in particular Theorem 2.7 still holds. Unguarded sum would require more machinery instead, in the line of [12,28]. However, as stated in the Introduction, recursion and sum can be represented in our fragment; in [21] Milner shows how recursive definitions can be expressed with replication: if we have $A(x) \stackrel{\text{def}}{=} P$, we replace every recursive call $A(y)$ in P with the subagent $\bar{a}y.0$ (a is a new name), obtaining an agent $\hat{P}$; so $!a(x).\hat{P}$ corresponds to $A(x)$ and if the call $A(z)$ appears is some agent, it can be replaced with $(\nu a)(\bar{a}z.0 \mid !a(x).\hat{P})$. Similarly, if $A(x) \stackrel{\text{def}}{=} P + Q$, each call $A(z)$ in an agent can be replaced by $(\nu a)(\bar{a}z.0 \mid !a(x).\hat{P} \mid !a(x).\hat{Q})$, so the "call" $\bar{a}z$ can choose one of the components $\hat{P}\{z/x\}$ and $\hat{Q}\{z/x\}$.

Different concurrent semantics for the same language would also be possible. On the more concurrent side, we can think of relaxing the sequencing power of prefixes by allowing the *unw* function to decompose them. More interesting would be to try to decompose replication as well, with the aim of achieving a level of parallelism comparable with the concurrent semantics of λ-calculus.

On the less concurrent side, we can eliminate the possibility of extruding the same channel in parallel. It is enough to introduce positive and negative pebbles to mark both the globality and the non-globality of names. The required changes are as follows:

$$unw((\nu x)P) = (x)(unw(P) \oplus \not\downarrow x)$$

$$\text{OUT1: } \bar{x}y.P \oplus \not\downarrow y \underset{\downarrow x}{\longrightarrow} unw(P) \oplus \downarrow y \qquad \text{OUT2: } \bar{x}y.P \underset{\downarrow x \oplus \downarrow y}{\longrightarrow} unw(P).$$

Different concurrent semantics can be obtained by changing the observations: the semantics of [2] can be expressed by considering $|G|$ in Definition 3.1 as the set of the arcs of G corresponding to sequential processes: so the dependencies between events deriving from the pebbles are not considered. Very recently Degano and Priami [14] have proposed a parametric approach to the concurrent semantics for the π-calculus; they have encoded the proofs of the transitions in their labels and have defined various observation functions, that permit to consider various forms of dependencies in the π-calculus. We believe that all these forms of dependencies can be defined in our context as well, by defining suitable observations over derivations (the advantage of the observations in Definition 3.1 is that they have a general meaning, i.e., they are not specific for the π-calculus). Instead, we think that the semantics presented in

this paper, which allows two processes to extrude the same name in parallel, as shown in the Introduction, cannot be easily matched in the other approaches. In addition, only our approach is equipped with an operational concurrent semantics, which makes explicit the amount of parallelism available.

In this work we considered only early semantics for the π-calculus, but it is not difficult to handle also the late case. However, if we want to keep the ordinary bisimulation of CCS we employed here, in contrast with the quite elaborate notion used for ordinary late equivalence, we need (in a similar fashion as [16]) to split every input step into two parts: the first makes a nondeterministic choice, if any, and the second chooses the input value:

$$\text{IN-: } x(y).P \underset{\downarrow x}{\longrightarrow} \lambda y.P$$

$$\text{IN1: } \lambda y.P \rightarrow (y)(unw(P) \oplus \downarrow y) \qquad \text{IN2: } \lambda y.P \underset{\downarrow z}{\longrightarrow} unw(P)\{z/y\}.$$

However, in order to keep the ordinary late semantics when the observation is interleaving, apparently it is necessary to force atomicity of the two parts of any input step. This condition is not required in the partial ordering case, which thus also in this case looks more natural.

Our last comment is about a comparison of the mechanisms used by π-calculus and by double-pushout rules to allow multiple use of the same resource, i.e., replication and reading mode. While the former is quite well studied (e.g., in the context of linear logic) we believe that the latter is more general. In fact, while a replicable resource must always be used in reading mode, so to say, a non replicable resource can be either read or consumed at will of its user. Here we propose an extension we call ROPI (for π-calculus with Replicating Output) where a new output prefix $\bar{\bar{x}}y.P$ is introduced, which duplicates the input agent while synchronizing:

$$\text{SYNC1: } \bar{\bar{x}}z.P \underset{x(y).Q}{\longrightarrow} unw(P) \oplus unw(Q)\{z/y\}.$$

References

[1] G. Berry and G. Boudol. The chemical abstract machine. In *Proc. POPL*, ACM, New York, 1990.

[2] M. Boreale and D. Sangiorgi. A fully abstract semantics of causality in the π-calculus. Tech. Rep. ECS-LFCS-94-297, University of Edinburgh, 1994. To appear in *Proc. STACS'95*.

[3] G. Boudol and I. Castellani. Flow models of distributed computations: three equivalent semantics for CCS. *Information and Computation*, 114:247–314, 1994.

[4] G. Boudol, I. Castellani, M. Hennessy and A. Kiehn. Observing localities. Tech. Rep. 4/91, University of Sussex, 1991. An extended abstract appeared in *Proc. MFCS'91*, LNCS 520 Springer Verlag. Also *Theoretical Computer Science*, 114:31–61, 1993.

[5] A. Corradini, H. Ehrig, M. Löwe, U. Montanari and F. Rossi. Abstract graph derivations in the double pushout approach. In: Hans Jürgen Schneider and Hartmut Ehrig, Eds., *Graph Transformations in Computer Science*, LNCS 776, pages 104–118. Springer Verlag, 1994.

[6] A. Corradini, H. Ehrig, M. Löwe, U. Montanari and F. Rossi. An event structure semantics for safe graph grammars. In Ernst-Rüdiger Olderog, Ed., *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56, pages 423–444. North Holland, 1994.

[7] A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoretical Computer Science*, 103:51–106, 1992.

[8] A. Corradini, U. Montanari and F. Rossi. An abstract machine for concurrent modular systems: CHARM. *Theoretical Computer Science*, 122:165–200, 1994.

[9] A. Corradini, U. Montanari and F. Rossi. Graph processes. Submitted for publication.

[10] Ph. Darondeau and P. Degano. Causal trees. In *Proc. ICALP'89*, LNCS 372. Springer Verlag, 1989.

[11] P. Degano, R. De Nicola and U. Montanari. Observational congruences for concurrency models. In *Proc. IFIP TC2 Workshop on Formal Description of Programming Concepts IV*, pages 105–132. North-Holland, 1987.

[12] P. Degano, R. De Nicola and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Inf.*, 26(1/2):59–91, Oct. 1988.

[13] P. Degano, J. Meseguer and U. Montanari. Axiomatizing net computations and processes. In *Proc. 4th Symp. on Logics in Computer Science*, IEEE 1989, pages 175–185.

[14] P. Degano and C. Priami. Causality for mobile processes. To appear in *Proc. ICALP'95*.

[15] H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. In *Proc. 3rd International Workshop on Graph Grammars and their Application to Computer Science*, LNCS 291, pages 3–14. Springer Verlag, 1987.

[16] G. Ferrari, U. Montanari and P. Quaglia. The weak late π-calculus semantics as observation equivalence. To appear on *Proc. CONCUR'95*.

[17] U. Goltz and W. Reisig. The nonsequential behavior of Petri nets. *Information and Computation*, 57:125–147, 1983.

[18] R. Gorrieri and U. Montanari. A simple calculus of nets. In J. Baeten and J. Klop, Eds., *Proc. CONCUR'90*, LNCS 458, pages 2–30. Springer Verlag, 1990. Also *Theoretical Computer Science*, to appear.

[19] C. Laneve and U. Montanari. Axiomatizing permutation equivalence in the λ-calculus. In 3^{th} *Int. Conf. on Algebraic and Logic Programming*, LNCS 632, pages 350–363. Springer Verlag, 1992.

[20] J. Meseguer. Conditional rewriting systems. *Theoretical Computer Science*, 96:73-155, 1992.

[21] R. Milner. The polyadic π-calculus: a tutorial. Tech. Rep. ECS-LFCS-91-180, University of Edinburgh, 1991. Also in F. L. Bauer, W. Brauer and H. Schwichtenberg, Eds., *Logic and Algebra of Specification*, NATO ASI Series F, Vol. 94. Springer Verlag, 1993.

[22] R. Milner. Pi-nets: a graphical form of π-calculus. In *Proc. ESOP'94*, LNCS 788, pages 26–42. Springer Verlag, 1994.

[23] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100:1–77, 1992.

[24] R. Milner, J. Parrow and D. Walker. Modal logic for mobile processes. In *CONCUR'91*, LNCS 527, pages 45–60. Springer Verlag, 1992.

[25] U. Montanari and F. Rossi. Graph grammars as context dependent rewriting systems: a partial ordering semantics. In *Proc. CAAP'92*, LNCS 582, pages 232–247. Springer Verlag, 1992.

[26] U. Montanari and F. Rossi. Contextual Nets. *Acta Informatica*, 1994, to appear.

[27] U. Montanari and D. Yankelevich. A parametric approach to localities. In *Proceedings* 19^{th} *ICALP*, LNCS 623, pages 617–628. Springer Verlag, 1992. Full paper in *Theoretical Computer Science*, to appear.

[28] E.-R. Olderog. Operational Petri net semantics for CCSP. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, LNCS 266, pages 196–223. Springer Verlag, 1987.

[29] J. Parrow. Interaction diagrams. To appear in *Proc. REX93*.

[30] D. Sangiorgi. *Expressing mobility in process algebras: first-order and higher-order paradigms*. PhD thesis CST-99-93, University of Edinburgh, 1992.

[31] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. Tech. Rep. ECS-LFCS-94-282, University of Edinburgh, 1994. An extract appeared in *Procs. TACS'94*.

[32] G. Schied. On relating rewriting systems and graph grammars to event structures. In: Hans Jürgen Schneider and Hartmut Ehrig, Eds., *Graph Transformations in Computer Science*, LNCS 776, pages 326–340. Springer Verlag, 1994.

[33] D. Taubner. *Finite representation of CCS and TCSP programs by automata and Petri nets*, LNCS 369. Springer Verlag, 1989.

[34] D. Walker. Objects in the π-calculus. *Information and Computation*, 1994. To appear.

[35] G. Winskel. An Introduction to Event Structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 364–397. Springer Verlag, 1989.